B13

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H01L 21/00, G05B 19/418 — A1

(11) International Publication Number: WO 98/57358

(43) International Publication Date: 17 December 1998 (17.12.98)

(21) International Application Number: PCT/US98/11320

(22) International Filing Date: 8 June 1998 (08.06.98)

(30) Priority Data:
08/871,746 9 June 1997 (09.06.97) US

(71) Applicant: APPLIED MATERIALS, INC. [US/US]; 3050 Bowers Avenue, Santa Clara, CA 95054 (US).

(72) Inventor: JEVTIC, Dusan; 444 Saratoga Avenue #34B, Santa Clara, CA 95050 (US).

(74) Agents: MOSER, Raymond, R., Jr. et al.; Applied Materials, Inc., P.O. Box 450A, Santa Clara, CA 95052 (US).

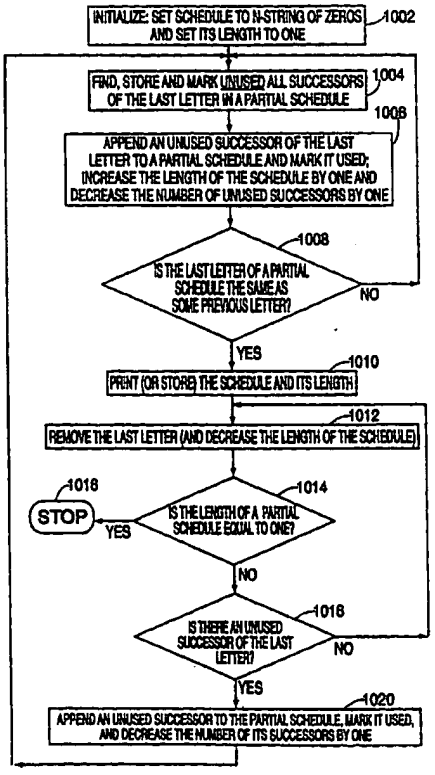
(81) Designated States: JP, KR, SG, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR AUTOMATICALLY GENERATING SCHEDULES FOR WAFER PROCESSING WITHIN A MULTICHAMBER SEMICONDUCTOR WAFER PROCESSING TOOL

(57) Abstract

A method and apparatus for producing schedules for a wafer in a multichamber semiconductor wafer processing tool comprising the steps of providing a trace defining a series of chambers that are visited by a wafer as the wafer is processed by the tool; initiatizing a sequence generator with a value of a variable defining initial wafer positioning within the tool; generating all successor variables for the initial variable value to produce a series of values of the variable that represent a partial schedule; backtracking through the series of variables to produce further partial schedules; and stopping the backtracking when all possible variable combinations are produced that represent all possible valid schedules for the trace. All the possible schedules; are analyzed to determine a schedule that produces the highest throughput of all the schedules.



METHOD AND APPARATUS FOR AUTOMATICALLY GENERATING SCHEDULES FOR WAFER PROCESSING WITHIN A MULTICHAMBER SEMICONDUCTOR WAFER PROCESSING TOOL

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a multiple chamber wafer processing tool and, more particularly, to a method and apparatus for automatically generating a schedule(s) for a semiconductor wafer within a multiple chamber semiconductor wafer processing tool.

2. Description of the Background Art

Semiconductor wafers are processed to produce integrated circuits using a plurality of sequential process steps. These steps are performed using a plurality of process chambers. An assemblage of process chambers served by a wafer transport robot is known as a multiple chamber semiconductor wafer processing tool or cluster tool. FIG. 1 depicts, in part, a schematic diagram of an illustrative cluster tool known as the Endura® System manufactured by Applied Materials, Inc. of Santa Clara, California.

The cluster tool 100 contains, for example, four process chambers 104, 106, 108, 110, a transfer chamber 112, a preclean chamber 114, a buffer chamber 116, a wafer orienter/degas chamber 118, a cooldown chamber 102, and a pair of loadlock chambers 120 and 122. Each chamber represents a different stage or phase of semiconductor wafer processing. The buffer chamber 116 is centrally located with respect to the loadlock chambers 120 and 122, the wafer orienter/degas chamber 118, the preclean chamber 114 and the cooldown chamber 102. To effectuate wafer transfer amongst these chambers, the buffer chamber 116 contains a first robotic transfer mechanism 124. The wafers 128 are typically carried from storage to the system in a plastic transport cassette 126 that is placed within one of the loadlock chambers 120 or 122. The robotic transport mechanism 124 transports the wafers 128, one at a time, from the cassette 126 to any of the three chambers 118, 102, or

A wafer's trace is the trajectory of a particular wafer through the cluster tool; that is, a trace is the order in which chambers are visited by a wafer (not necessarily C_{i+1} after C_i). This should be distinguished from the term "processing sequence" which is the order of applying processes (recipes) to a wafer. If more than one chamber performs the same process (parallel chambers), a given processing sequence may be satisfied by several different traces.

A wafer which completes its processing sequence and is returned to the loadlock is said to be processed by the tool. Roughly speaking, a tool's throughput is the number of wafers processed by the tool per unit of time. That is, if the tool needs t seconds to process n_t wafers, then

$$S_t := \frac{n_t}{t} \tag{3}$$

is the tool's throughput measured in the interval [0,t].

There are many ways to improve the tool's throughput for a given processing sequence. However, one important improvement is to use efficient scheduling routines for a given processing sequence.

The optimization of scheduling involves the choice of criteria used in deciding when to transfer a wafer from one chamber into the next (and which wafers should be moved, if any, prior to that move). A routine which schedules the movement of wafers through the cluster tool (based on a given processing sequence) is referred to as a "scheduling routine."

The steady-state throughput of a tool under scheduling routine A is denoted by S(A). If n>1 then, depending on a given processing sequence, one may consider a number of scheduling routines that fulfill the processing sequence. The routine which maximizes the value of throughput is deemed the "optimum" routine and the maximum attainable value of throughput is known as the tool's "capacity." That is, if A is the set of all possible scheduling routines for a given processing sequence, then A* is optimum if

$$S(A^*)=\max\{S(A) \mid A \in A\} \tag{4}$$

successor letters, as well as the total number of successors for the input letter. The invention provides individual "modules" for successor computation for serial traces, parallel traces, and mixed traces. Using a backtracking technique to repeatedly compute, from any letter, all possible successor letters, and then compute all possible successor letters of the successor letters, a schedule tree is derived. The schedule tree contains all possible schedules that will fulfill a given trace. Each and every schedule can then be modeled to determine the expected throughput of each schedule. By comparing the throughput associated with each schedule, an optimal schedule or schedules is identified.

BRIEF DESCRIPTION OF THE DRAWINGS

The teachings of the present invention can be readily understood by considering the following detailed description in conjunction with the accompanying drawings, in which:

Fig. 1 depicts a schematic diagram of a multiple chamber semiconductor wafer processing tool being controlled by a sequencer that operates using scheduling routines generated by a schedule generator in accordance with the present invention;

Fig. 2 depicts block diagram of schedule generator that performs operative steps in accordance with the present invention;

Fig. 3 depicts a flow diagram of a 4-chamber serial trace;

Fig. 4 depicts a flow diagram of a 4-chamber mixed trace;

Fig. 5 depicts a flow diagram of a schedule optimization routine of the present invention;

Fig. 6 depicts a tree diagram representing all possible schedules for a 2-chamber serial trace;

FIG. 6A depicts a schematic diagram of a 2-chamber serial trace of FIG. 6 showing a wafer in position (1,0);

Fig. 7 depicts a tree diagram representing all possible schedules for a 3-chamber serial trace;

The microprocessor 200 cooperates with conventional support circuitry 206 such as power supplies, clock circuits, cache, and the like as well as circuits that assist in executing the software routines. As such, it is contemplated that some of the process steps discussed herein as software processes may be implemented within hardware, e.g., as circuitry that cooperates with the microprocessor to perform various process steps. The schedule generator 50 also contains input/output circuitry 208 that forms an interface between conventional input/output (I/O) devices 214 such as a keyboard, mouse, and display as well as an interface to the sequencer. Although the schedule generator 50 is depicted as a general purpose computer that is programmed to determine scheduling routines in accordance with the present invention, the invention can be implemented in hardware as an application specific intergrated circuit (ASIC). As such, the process steps described herein are intended to be broadly interpreted as being equivalently performed by software, hardware, or a combination thereof.

The automatic schedule generator 50 of the present invention executes a schedule generation routine 210 that generates all possible schedules for a given trace. A schedule optimization routine 212 facilitates an automated process of producing an optimum schedule for a given cluster tool using an exhaustive search of all possible schedules.

The following definitions are used throughout this disclosure:

"Tool configuration" describes physical placement of chambers within a cluster tool. For example, the tool may have chambers C_1, C_2, C_3 and C_4, a LoadLock (LL) as well as one or more robots.

"Process sequence" is the order in which processes are applied to a given wafer. For example, P_n is the name of the n-th process (e.g., etch) and, P_1, P_2, P_3, (which also may be written as $P_1 \rightarrow P_2 \rightarrow P_3$) is a process sequence.

"Processing capability" of a cluster tool is the result of mapping a required process sequence onto the set of chambers within the tool. The image of this mapping is

which starts and ends with the same letter (e.g., x), this is the only repeated letter, and a successor v of a given letter u must satisfy alphabet dependent rules, i.e., rules which define a valid trace.

Traces are available in three different configurations, A trace is a parallel trace if it is comprised of exactly one stage; a trace is a serial trace if each stage has exactly one chamber and a trace is a mixed trace if it is neither serial nor parallel. (Clearly, to have a mixed trace, the number of chambers in the trace is at least three.) A trace is said to be knotted if there is a chamber whose name appears more than once in the trace (that is, the corresponding process sequence contains a processing loop). To illustrate, Figs. 3 and 4 schematically depict 4-stage serial and mixed traces, respectively.

Fig. 5 depicts a high level flow diagram of the schedule optimization routine 212. The optimization routine contains a schedule generation routine 210 that produces all possible schedules in an alphabet induced by a given trace. Routine 212 is an automated process that performs the following steps:

a) Input a trace L (step 500),

b) Produce all possible schedules over L (routine 210) using a two step process, where the first step (step 508) generates all possible successor positions (letters) to which a wafer can be moved from a present position (letter) and the second step (step 510) uses a backtracking technique to change wafer positions such that other successor positions (letters) can be computed by step 508,

c) Evaluate each of the schedules in (b) with respect to throughput (for a given set of robot and process parameters) (step 504),

d) Record a schedule or a set of schedules which have the highest throughput for the given trace L (step 506).

Since step (c) requires a throughput simulation program, for computational efficiency, steps (a), (b) and (d) are generally incorporated into the simulation program.

s_3) If, for some $k \notin \{0,n\}$, $\bar{u}[k]=1$ and $\bar{u}[k+1]=0$, then $\bar{v}[k]=0$ and $\bar{v}[k+1]=1$. For all $i \notin \{k,k+1\}$, $\bar{v}[i]=\bar{u}[i]$. (This corresponds to a wafer being moved from C_k into C_{k+1}.)

FIG. 6 illustrates all possible schedules available (i.e., two schedules) in a 2-chamber serial trace. FIG. 6A depicts a schematic diagram of the 2-chamber serial trace of FIG. 6 having a wafer in position represented by the 2-tuple (1,0). These n-tuples are referred to herein as the coordinates of wafer positioning. From position (1,0), the schedule of FIG. 6 dictates that the wafer is next moved to a position represented by the 2-tuple (0,1), i.e., a wafer is now in chamber C_2 and no wafer is in chamber C_1. Thereafter, the schedule may follow one of two paths, either the wafer in C_2 is moved to the loadlock (a wafer positioning that is represented by 2-tuple (0,0)) or another wafer is moved into chamber C_1 (a wafer positioning that is represented by 2-tuple (1,1)). As such, each 2-tuple represents a set of possible positions for a wafer or wafers that validly fulfill a step in the trace.

Similarly, FIG. 7, illustrates the seven possible schedules available in a 3-chamber serial trace and FIG. 7A depicts a schematic diagram of the trace of FIG. 7 having a wafer positioning represented by the 3-tuple (0,1,0). From Fig. 7, the strings

S_p=(1,1,1)(1,1,0)(1,0,1)(0,1,1,)(1,1,1)
S_w=(1,0,0)(0,1,0)(0,0,1)(0,0,0)(1,0,0)
S_x=(1,0,1)(0,1,1)(0,1,0)(1,1,0)(1,0,1).

represents particular scheduling routines that are generated by the schedule generator for a three chamber serial trace.

Such schedules may contain a set of robot and chamber parameters that yield higher or lower throughput than other schedules in the set of all schedules. As such, the only way to determine an optimum schedule is to examine the throughput under all possible schedules and, using the optimization routine, determine which of the schedules is optimal.

Generating all SerCount($\bar{u}$) successors of a given letter $\bar{u}$ is not particularly difficult. As each successor of $\bar{u}$ is generated, it is stored in a binary matrix Z that has SerCount($\bar{u}$) rows and (n+1) columns. The last column of Z is reserved for a Boolean variable that is set to true if the successor was used in a partial schedule and is set to false if the successor was not used. This entry is used later in the backtracking routine (discussed below with reference to FIG. 10) that generates all possible schedules for a given trace. The successors of a given letter are determined by the following function.

```
function SerGenerator(ū:letter): matrix;
var
   i:integer;
begin
   if ū[1] = 0 then begin
      copy (ū, v̄);
      v̄[1] = 1;
      store(v̄, Z);
   end;
   if ū[n] = 1 then begin
      copy (ū, v̄);
      v̄[n] = 0;
      store (v̄, Z);
      end;
      for i:=1 to n-1
      if ū[i]=1 and ū[i+1]=0 then begin
         copy (ū, v̄);
         v̄[i]=0; v̄[i+1]:=1;
         store (v̄, Z);
      end;
      return (Z);
end;
```

There are two functions which are used repeatedly in the above pseudo-code. Function *copy* ($\bar{u}$, $\bar{v}$) returns letter $\bar{u}$ that is a replica of letter $\bar{v}$. This manner of implementing rules (s_1), (s_2), and (s_3), in which the routine first copies $\bar{u}$ into $\bar{v}$ and then modifies $\bar{v}$, is not inefficient because $\bar{u}$ and $\bar{v}$ differ in at most two coordinates. Function *store* ($\bar{v}$, Z) copies letter $\bar{v}$ into a proper row of matrix Z. Note that in the above module, the

binary n-tuple. This is the only repeated letter in the word. In addition, if $\bar{v}$ is a successor of $\bar{u}$, then $\bar{u}$ and $\bar{v}$ differ in at most two coordinates and the following rules define the relationship of $\bar{u}$ and $\bar{v}$:

m_1) If for some $i \in F_1$, $\bar{u}[i]=0$, then $\bar{v}[i]=1$. For all $k \neq i$, $\bar{v}[k]=\bar{u}[k]$ (This corresponds to a wafer being moved from the loadlock into stage 1.)

m_2) If for some $i \in F_k$, $\bar{u}[i]=1$, then $\bar{v}[i]=0$. For all $j \neq i$, $\bar{v}[j]=\bar{u}[j]$. (This corresponds to a wafer being moved from the last stage F_k into the loadlock).

m_3) If for some $i \in F_t$ and some $j \in F_{t+1}$, $\bar{u}[i]=1$ and $\bar{u}[j]=0$, then $\bar{v}[i]=0$ and $\bar{v}[j]=1$. For all $r \notin \{i,j\}$, $\bar{v}[r]=\bar{u}[r]$. (This corresponds to a wafer being moved from stage F_t into the next stage F_{t+1}.)

In determining the number of successors of a given letter $\bar{u}$, it will be handy to define a sequence M_o=0 and

$$M_t = |F_1| + |F_2| + \cdots + |F_t|,$$

where $|F_t|$ is the size (number of chambers) of stage F_t. The above sequence reflects the partition of the index set of $\bar{u}$ into stages. Clearly, M_k=n, where n is the number of chambers. The number of successors of $\bar{u}$ is determined by the following function:

```
function MixCount(ū:letter):integer;
var
  t, i, j, nmb:integer;
begin
   nmb:=0
   for i:=1 to M1
      if ū[i]=0
      then nmb:=nmb+1
   for j:=1+Mk-1 to Mk
      if ū[i]=1
      then nmb:=nmb+1
   for t:=1 to k-1
```

```
        for i:=1+M_{t-1} to M_t
        for j:=1+M_t to M_{t+1}
          if ū[i]=1 and ū[j]=0 then begin
          copy (ū, v̄);
          v̄[i]:=0; v̄[j]:=1;
          store(v̄, Z)
        end;
        return (Z);
      end;
```

Functions *copy* $(\bar{u}, \bar{v})$ and *store* $(\bar{v}, Z)$ are the same as in the corresponding routine for serial traces. (Note that this time matrix Z has MixCount$\bar{u}$ rows and (n+1) columns.) Again, if M_t=t and k=n in the above function, then MixGenerator$(\bar{u})$ becomes SerGenerator$(\bar{u})$. For pure parallel traces, due to k=1, a function that generates successors of a given letter $\bar{u}$ is:

```
function ParGenerator(ū:letter):matrix;
var
   i:integer;
begin
   for i:=1 to n
   begin
      copy (ū, v̄);
      if ū[i]=1
      then v̄[i]:=0
      else v̄[i]:=1
      store (v̄, Z)
   end;
   return (Z)
end;
```

Note the similarity between functions that count successors and functions that generate successors. In fact, conditions for identifying a successor are identical in both types of function; the difference is what is performed once the condition is detected.

FIG. 8 depicts an illustrative schedule tree for a 3-chamber mixed trace, (e.g., $LL \rightarrow C_1 \rightarrow (C_2 \vee C_3) \rightarrow LL$), where the successors of a particular letter are determined using the MixGenerator$(\bar{u})$ pseudo-code. FIG. 8A depicts a schematic diagram of the trace of FIG. 8 having wafers positioned in position (1,1,0).

$$S = \overline{xz}\cdots\overline{uv}\cdots\overline{yx},$$

which starts and ends with the same letter and this is the only repeated letter. Furthermore, any two consecutive letters $\bar{u}$ and $\bar{v}$ (where $\bar{v}$ is a successor of $\bar{u}$) differ in at most three coordinates and are related in accordance with the following rules:

a) If $\bar{u}[1]=0$ and $\bar{u}[n+1]=0$, then $\bar{v}[1]=1$ and $\bar{v}[n+1]=1$. For all i $\notin\{1,n+1\}$, $\bar{v}[i]=\bar{u}[i]$. (This correspondence to a wafer being moved from the loadlock to C_1.)

b) If $\bar{u}[n]=1$ and $\bar{u}[n+1]=n$, then $\bar{v}[n]=0$ and $\bar{v}[n+1]=0$. For all i $\notin\{n,n+1\}$, $\bar{v}[i]=\bar{u}[i]$. (This corresponds to a wafer being moved from C_n into the loadlock.)

c) If for some r$\notin\{0,n\}$, $\bar{u}[r]=1$ and $\bar{u}[r+1]=0$ and $\bar{u}[n+1]=r$, then $\bar{v}[r]=0$ and $\bar{v}[r+1]=1$ and $\bar{v}[n+1]=r+1$. For all i $\notin\{r,r+1,n+1\}$, $\bar{v}[i]=\bar{u}[i]$. (This corresponds to a wafer being moved from C_k into $C_{(k+1)}$, where neither C_k nor $C_{(k+1)}$ is a loadlock.)

d) If $\bar{u}[1]=0$ and $\bar{u}[n+1]=j$ where $j \neq 0$, then $\bar{v}[n+1]=0$. For all $i \neq$ n+1, $\bar{v}[i]=\bar{u}[i]$. (This corresponds to a robot moving from home position at C_j to a home position at a loadlock in preparation for a wafer moving from the loadlock into C_1.)

e) If $\bar{u}[n]=1$ and $\bar{u}[n+1]=j$ where $j\neq n$, then $\bar{v}[n+1]=n$. For all $i \neq$ n+1, $\bar{v}[i]=\bar{u}[i]$. (This corresponds to a robot moving from a home position at C_j to a home position at C_n in preparation for a wafer move from C_n into loadlock.)

f) If for some $r \notin \{0,n\}$, $\bar{u}[r]=1$ and $\bar{u}[r+1]=0$ and $\bar{u}[n+1]=j$ where $j \neq r$, then $\bar{v}[n+1]=r$. For all $i\neq$ n+1, $\bar{v}[i]=\bar{u}[i]$. (This corresponds to a robot moving

Let $\bar{u}_1$ be the starting letter of a schedule. By using the rules for adding a successor letter as discussed in Sections B, C or D above, the foregoing routines build a partial schedule, say $S=\bar{u}_1\bar{u}_2\cdots\bar{u}_k$. There are two questions to answer every time a new letter $\bar{u}_{k+1}$ is added to partial trace S:

a) Is $\bar{u}_1\bar{u}_2\cdots\bar{u}_k\bar{u}_{k+1}$ a full schedule?

b) If $\bar{u}_1\bar{u}_2\cdots\bar{u}_k\bar{u}_{k+1}$ is a full schedule, are there other full schedules which have not been recorded?

A word $\bar{u}_1\bar{u}_2\cdots\bar{u}_{k+1}$ is recognized as a full schedule if it is built according to rules for successor letters and if there exists an index $i<k+1$ such that $\bar{u}_i=\bar{u}_{k+1}$ and all letters $\bar{u}_1\ \bar{u}_2\cdots\bar{u}_k$ are different. Thus, to determine a full schedule a routine checks whether or not

$$\bar{u}_{k+1}\neq\bar{u}_i, \qquad i=1,2,\cdots,k,$$

for every newly appended letter $\bar{u}_{k+1}$ which is a proper successor of $\bar{u}_k$.

Once it is found that $\bar{u}_i=\bar{u}_{k+1}$ for some $i<k+1$, the routine either prints or stores the full schedule $\bar{u}_i\ \bar{u}_{i+1}\cdots\bar{u}_{k+1}$. To find other schedules, the routine removes $\bar{u}_{k+1}$ from the full schedule S and looks at some other unused successor of $\bar{u}_k$. If there is such a successor, say letter $\bar{z}$, the routine checks if $\bar{u}_1\ldots\bar{u}_k\bar{z}$ is a full schedule. If $\bar{u}_1\ldots\bar{u}_k\bar{z}$ is not a full schedule, the routine looks at unused successors of $\bar{z}$ and so on. If $\bar{u}_1\ldots\bar{u}_k\bar{z}$ is a full schedule, the routine removes $\bar{z}$ and looks at another unused successor of $\bar{u}_k$. If there are no unused successors of $\bar{u}_k$, the routine goes back (backtrack) and looks at unused successors of $\bar{u}_{k-1}$ and so on, until the routine returns to the starting letter $\bar{u}_1$. Basically, the routine contains the following sequence of steps:

only one successor ($\bar{e}_1+\bar{e}_3$). Hence, $\bar{0}\bar{e}_1\bar{e}_2(\bar{e}_1+\bar{e}_2)(\bar{e}_1+\bar{e}_3)$ and $\bar{0}\bar{e}_1\bar{e}_2\bar{e}_3(\bar{e}_1+\bar{e}_3)$ and $\bar{0}\bar{e}_1\bar{e}_2\bar{e}_3\bar{e}_4$ are all partial schedules and so on.

As mentioned previously, letters comprising a given word (or partial schedule) S are distinguished by their positions; so S(1) is the first letter in S, S(2) is the second, and so on. The level of a search tree is denoted by L this is also the length of a partial schedule. Partial schedules of length L are thus paths of length L in a search tree. Function scnt($\bar{x}$) returns the number of successors of $\bar{x}$. As such *scnt*($\bar{x}$) is either SerCount($\bar{x}$) or MixCount($\bar{x}$) or either of these for a model which includes robot movements. If S is a partial schedule of length L, then S+w or Sw is a partial schedule of length L+1 and S(L+1)=w. Similarly, if length of S is L and S(L)=w, then S-w has length L-1 (in short, + means append a letter and - means remove a letter).

Finally, a commitment to data organization (structure) is necessary in an efficiently designed routine. Keep in mind that the number of chambers (and thus the number of successors of a given word) is relatively small. Thus, it does not make any difference if the routine generates all successors of a given letter and stores them, as opposed to dynamically generating the successors one-by-one as the need arises.

A basic schedule generator routine can be summarized by the following five steps:

1. Initialize the schedule: S←$\bar{0}$ and L←1 and go to Step 2.
2. $\bar{x}$←S(L) and δ←cnt($\bar{x}$). Store δ successors of $\bar{x}, \bar{y}_1, \bar{y}_2, \ldots, \bar{y}_\delta$, and mark them unused. Go to Step 3.
3. S←S+$\bar{y}_1$ and L←L+1 and δ←δ-1. Mark $\bar{y}_1$ used and go to Step 4.
4. Compare S(L) with S(1), S(2),...,S(L-1), respectively. If S(i)=S(L) for some i<1, print S and L and go to Step 5; else, go to Step 2.

routine prints (stores) the schedule; else, the routine continues with building the schedule.

When a partial schedule becomes a full schedule, $S(1)S(2)...S(L)$, after storing the schedule, the routine removes the last letter $S(L)$ and look for some other unused successor of $S(L-1)$. If there are some unused successors, the routine appends a successor to the partial schedule, finds its successors, appends one of these successors and so on. If there are no successors, the routine removes $S(L-1)$ from S and looks for unused successors of $S(L-2)$ and so on. The program terminates when $L=1$ and $\delta=0$ (meaning there are no unused successors of the first letter).

The routine above is valid for any representation of the scheduling problem. That is, either serial or mixed traces with letters from $\{0,1\}^n$ or either of these traces with robot position being part of the model (and thus alphabet from $\{0,1\}^n \times \{0,1...,n\}$). Clearly, functions that count and generate successors of a given letter are different each time.

Fig. 10 depicts a flow diagram of a schedule generation routine 1000 that operates as generally discussed above. The routine 1000 begins at step 1002 by initializing the schedule, e.g., setting an initial letter to an n-tuple (n-string) of zeros. At step 1004, the routine finds, stores and marks all unused successors of the last letter in a partial schedule. The successor letters are determined using the pseudo-code routines SerGenerator, ParGenerator and MixGenerator and the number of successor letters for each letter is determined using SerCount, ParCount and MixCount. Of course, as mentioned above, if the robot position is to be taken into account, these pseudo-code routines must be appropriately modified to accommodate the expanded letters and the modify rules of successor generation.

Then, at step 1006, the routine appends an unused successor of the last letter to a partial schedule as well as increases the length of the schedule by one and decreases the number of unused successors by one. The routine

Although various embodiments which incorporate the teachings of the present invention have been shown and described in detail herein, those skilled in the art can readily devise many of the varied embodiments that still incorporate these teachings.

6. The method of claim 4 wherein each letter contains a n-tuple, where n represents a total number of wafer positions within said trace.

7. The method of claim 6 wherein n represents a total number of wafer positions within said trace plus a transport robot position.

8. The method of claim 3 further comprising the step of:

(f) identifying at least one schedule in all possible valid schedules that provides an optimal throughput for the trace.

9. The method of claim 8 wherein said identifying step (f) further comprises the step of computing the throughput for each and every schedule of said all possible schedules and selecting an optimum schedule for the trace which produces the highest throughput.

10. The method of claim 3 wherein said backtracking step (d) further comprises the steps of:

(a') initializing a schedule to an n-string of zeros having a length of one;

(b') identifying all unused successors of a last letter in a partial schedule;

(c') appending an unused successor to said partial schedule;

(d') increasing the length of the partial schedule by one and decreasing a number of unused successors by one;

(e') querying whether the last letter of the partial schedule is the same as a previous letter in the partial schedule and, if not, repeat steps (b'), (c'), (d') and (e') until the last letter of the partial schedule is the same as a previous letter in the partial schedule;

(f') storing the partial schedule as a full schedule and storing the length of the full schedule;

(g') removing the last letter of the full schedule to produce a new partial schedule and decreasing the length of the full schedule by one;

which produces further values of said variable representing additional series of values that represent additional partial schedules, whereby all possible schedules for said trace are generated.

16. The apparatus of claim 15 further comprising a throughput model, coupled to said sequence generator, for computing a throughput value for each possible schedule in said all possible schedules.

17. The apparatus of claim 16 further comprising means for identifying at least one schedule that has the largest throughput value.

18. A schedule data structure, stored in a computer readable storage medium, of a schedule for processing a wafer in a multichamber semiconductor wafer processing tool comprising:

a plurality of n-tuples, where n is a total number of possible wafer positions, and the plurality of n-tuples contains only one duplicate n-tuple.

19. The schedule data structure of claim 18 wherein n is a total number of possible wafer positions plus a position of transport robot position.

20. The schedule data structure of claim 18 wherein the n-tuple has a form $(x_1, x_2, x_3, \ldots, x_n)$, where x defines the contents of a particular chamber at a particular point in a schedule and x has a value 0 when a wafer is not positioned in the chamber and a value of 1 when a wafer is positioned in the chamber.

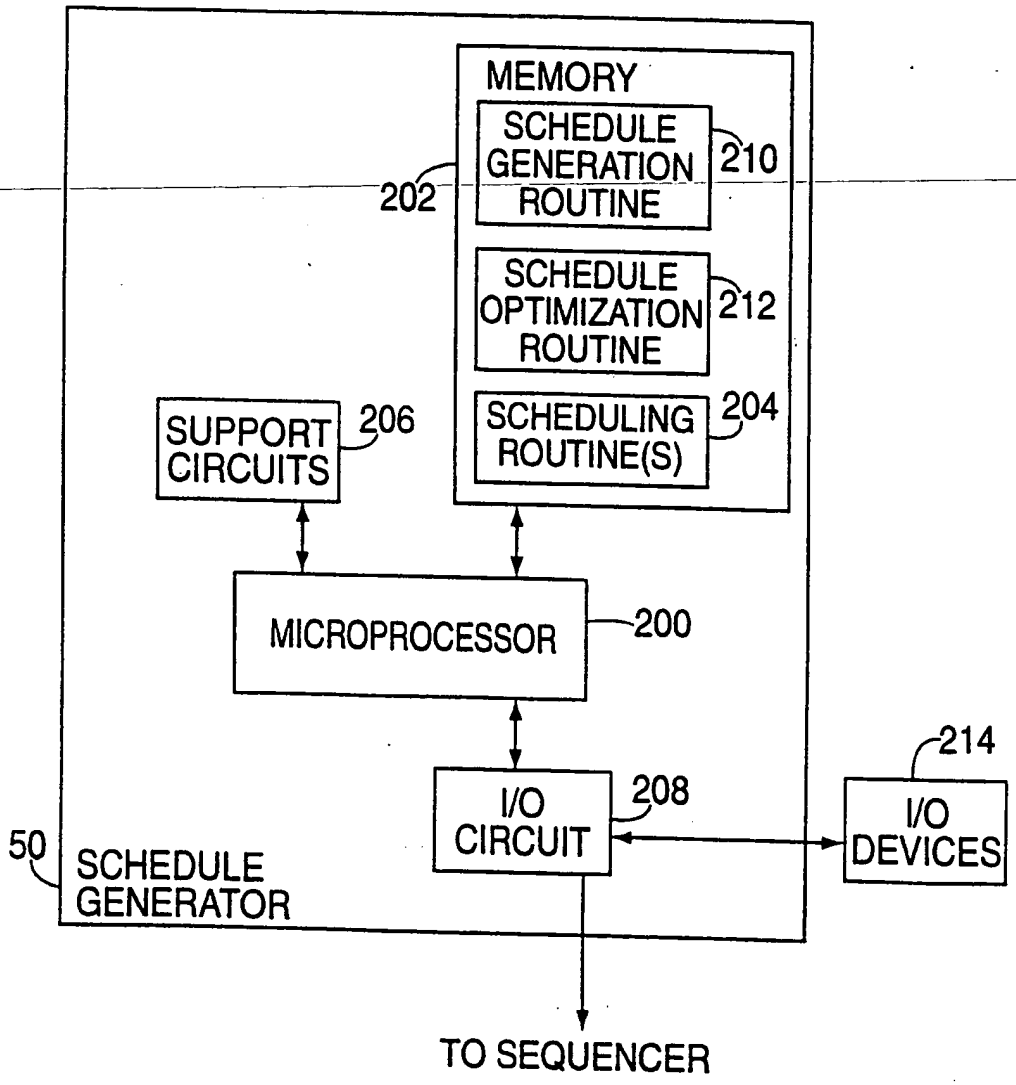

MEMORY
SCHEDULE GENERATION ROUTINE
210
202
SCHEDULE OPTIMIZATION ROUTINE
212
SCHEDULING ROUTINE(S)
204
SUPPORT CIRCUITS
206
MICROPROCESSOR
200
I/O CIRCUIT
208
214
I/O DEVICES
50
SCHEDULE GENERATOR
TO SEQUENCER

FIG. 2

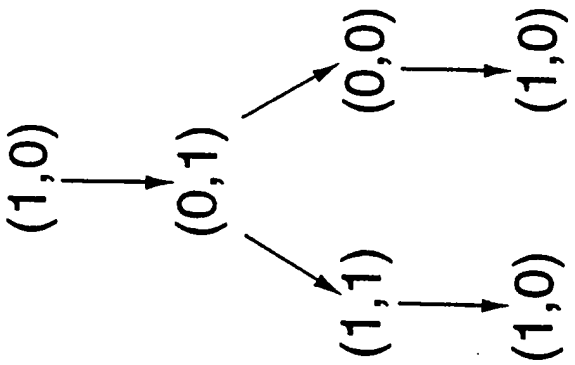


FIG. 6

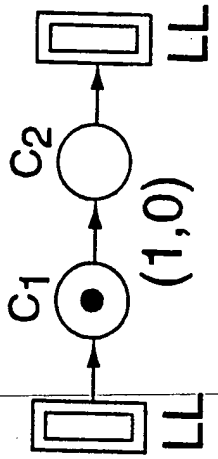


FIG. 6A

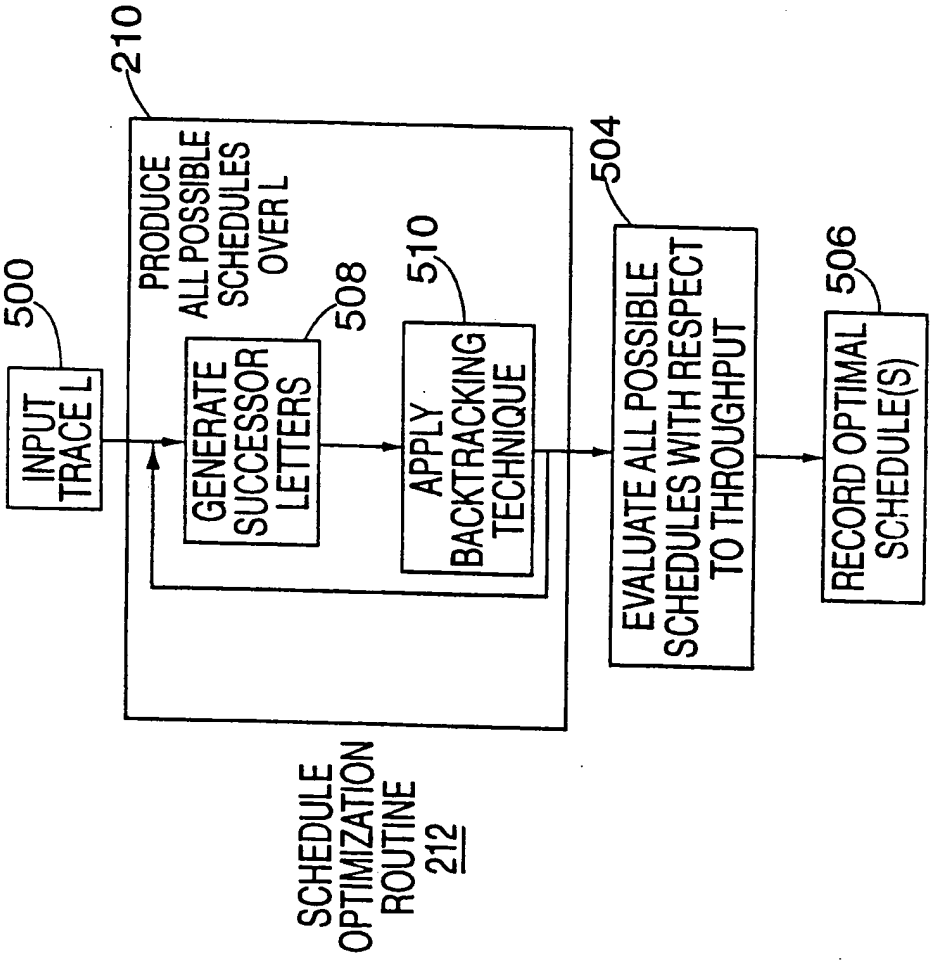


FIG. 5

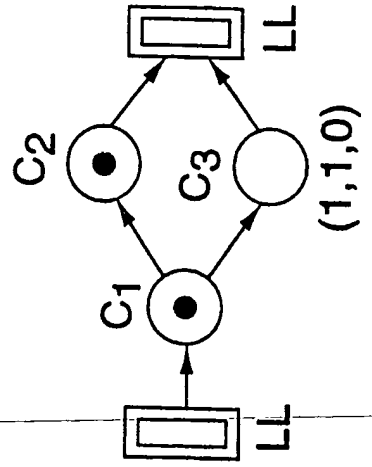

LL
C2
C3
(1,1,0)
C1
LL

FIG. 8A

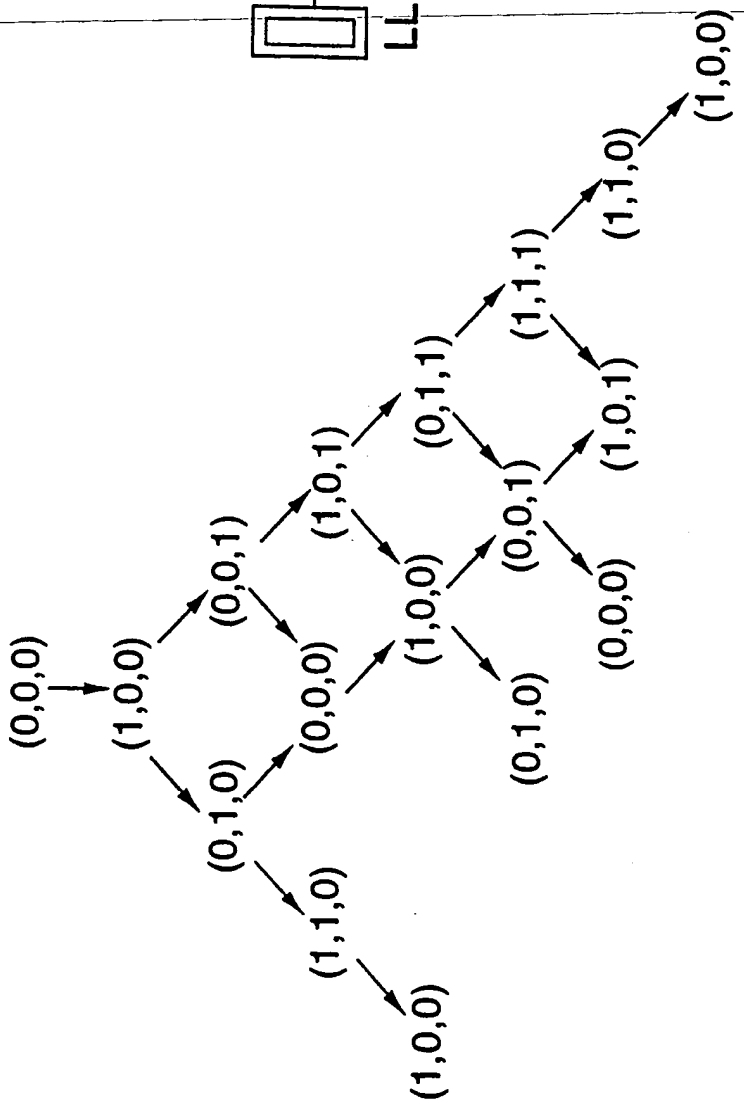

(0,0,0)
(1,0,0)
(0,1,0)
(1,1,0)
(1,0,0)
(0,0,1)
(0,0,0)
(1,0,1)
(1,0,0)
(0,1,0)
(0,1,1)
(0,0,1)
(0,0,0)
(1,1,1)
(1,0,1)
(1,1,0)
(1,0,0)

FIG. 8

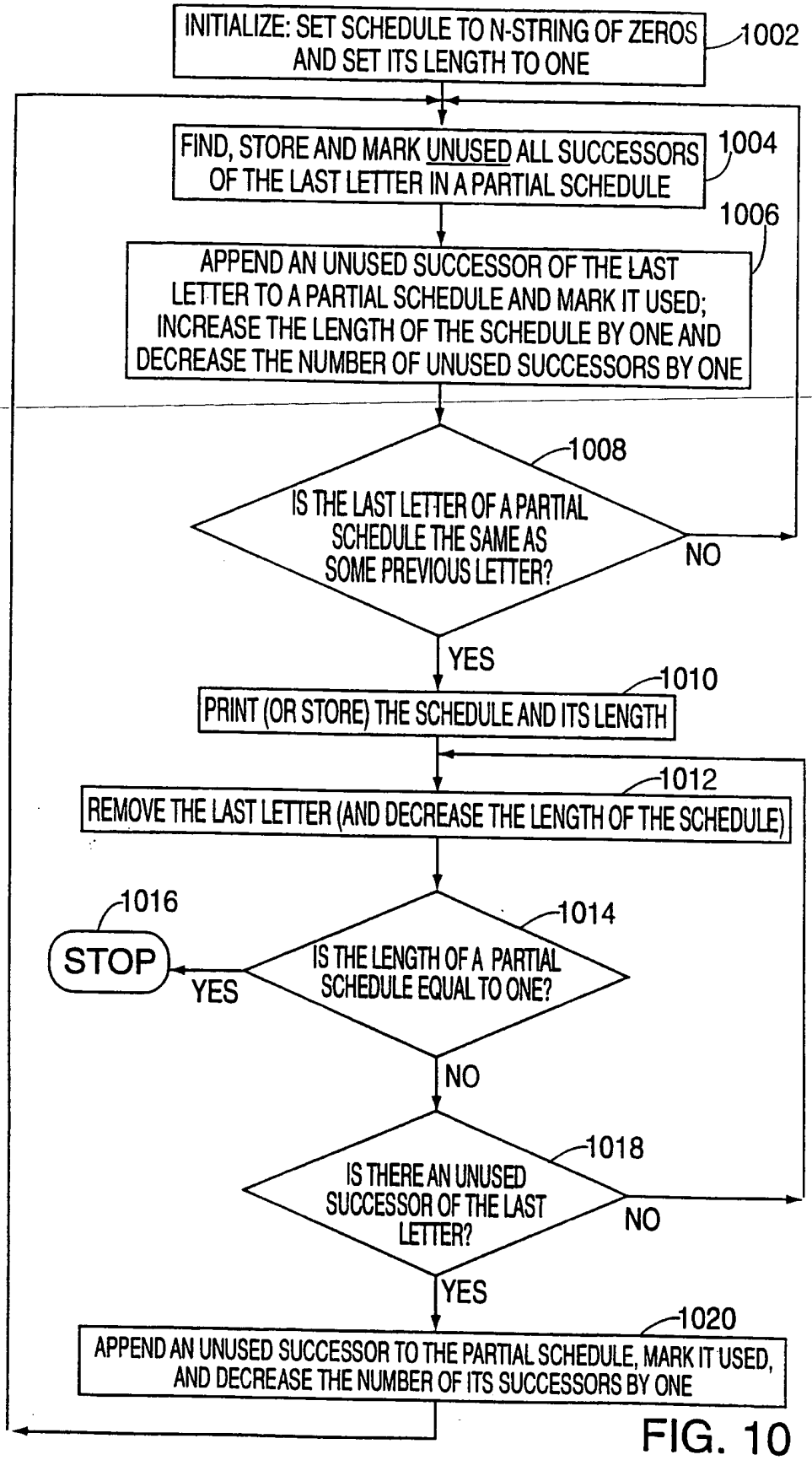

INITIALIZE: SET SCHEDULE TO N-STRING OF ZEROS AND SET ITS LENGTH TO ONE
1002
FIND, STORE AND MARK UNUSED ALL SUCCESSORS OF THE LAST LETTER IN A PARTIAL SCHEDULE
1004
APPEND AN UNUSED SUCCESSOR OF THE LAST LETTER TO A PARTIAL SCHEDULE AND MARK IT USED; INCREASE THE LENGTH OF THE SCHEDULE BY ONE AND DECREASE THE NUMBER OF UNUSED SUCCESSORS BY ONE
1006
IS THE LAST LETTER OF A PARTIAL SCHEDULE THE SAME AS SOME PREVIOUS LETTER?
1008
NO
YES
PRINT (OR STORE) THE SCHEDULE AND ITS LENGTH
1010
REMOVE THE LAST LETTER (AND DECREASE THE LENGTH OF THE SCHEDULE)
1012
IS THE LENGTH OF A PARTIAL SCHEDULE EQUAL TO ONE?
1014
STOP
1016
YES
NO
IS THERE AN UNUSED SUCCESSOR OF THE LAST LETTER?
1018
NO
YES
APPEND AN UNUSED SUCCESSOR TO THE PARTIAL SCHEDULE, MARK IT USED, AND DECREASE THE NUMBER OF ITS SUCCESSORS BY ONE
1020

FIG. 10

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int tional Application No
PCT/US 98/11320

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 4896269 A	23-01-1990	NONE	